Experiment 5

AIM: To write a Python program for linear and circular convolution of two discrete time sequences. Plot all the sequences and verify the result by analytical calculation.

Objective:

1. To convolve two discrete sequences using linear convolution
2. To convolve two discrete sequences using circular convolution

Theory: Linear Convolution**:** For a system with the impulse response h[n] the output for any arbitrary input x[n] is given by convolution defined as,

$$y[n] = x[n] * h[n]$$

$$y[n] = \sum_{k=-\infty}^{\infty} x[k]h[n-k]$$

Circular convolution**:** It is periodic convolution. and are the two finite duration sequences of length N, the circular convolution between though sequence is given by

$$x_3[n] = x_1[n] \otimes x_2[n]$$

$$x_3[n] = \sum_{m=0}^{N-1} x_2[m]x_1[((n-m))_N]$$

The linear convolution of an N-point vector, x, and a L-point vector, y, has length N+L-1.For the circular convolution of x and y to be equivalent, you must pad the vectors with zeros to length at least N+L-1 before you take the DFT. After you invert the product of the DFTs, retain only the first N+L-1 elements.

Python Code

(a) #Program to perform linear convolution

```
import numpy as np
#impulse response
h = [1,2,3,3,2,1];
#input response
x = [1,2,3,4,5];
y = np.convolve(x,h,mode='full')
print('Linear convolution using NumPy built-in function output response y=\n',y)
```

#Results

Linear convolution using NumPy built-in function output response y= [1 4 10 19 30 36 35 26 14 5]

(b) # Python program to compute circular convolution of two arrays

```
import numpy as np
#impulse response
h = [1,2,3,4,5];
#input response
x = [1,2,1];
# Pad sequences to the same length
N=max(len(x), len(h))
x_padded = np.pad(x, (0, N-len(x), mode='constant')
h_padded= np.pad(h, (0, N-len(h), mode='constant')
# Perform circular convolution using np.fft.ifft()
X = np.fft.fft(x_padded)
H = np.fft.fft(h_padded)
Y = np.fft.ifft(X * H)
print("Circular Convolution Result:", np.real(Y))
# Result of circular convolution is [15  9  8 12 16]
```

Practice question:

1. Generate a sinusoidal signal using python and plot the same
2. Generate two sinusoidal signals having 400Hz and 4000Hz using python and plot the convolution of two signals.

Sl. No	Criteria	Max Marks	Marks obtained
Data sheet			
A	Problem statement	10	
B	Design & specifications	10	
C	Expected output	10	
Record			
D	Simulation/ Conduction of the experiment	15	
E	Analysis of the result	15	
	Viva	40	
	Total	100	
Scale down to 10 marks			

Experiment 6

AIM: To write a Python program for circular correlation of two discrete time sequences. Plot all the sequences and verify the result by analytical calculation.

Objective:

1. To do the autocorrelation of a discrete sequence
2. To do the cross correlation of two discrete sequences

Theory

The correlation operation measures the similarity between two sequences by computing the convolution of one sequence with a time-reversed version of the other sequence. There are two types of correlation: cross-correlation and auto-correlation.

Cross-correlation measures the similarity between two different sequences. Given two sequences, let's say x[n] and y[n], the cross-correlation is computed as:

$r_{xy}[k] = \sum(x[n] * y[n-k])$

where k is the lag or shift applied to the second sequence y[n]. The resulting sequence r_xy[k] represents the similarity between the two sequences at each lag k.

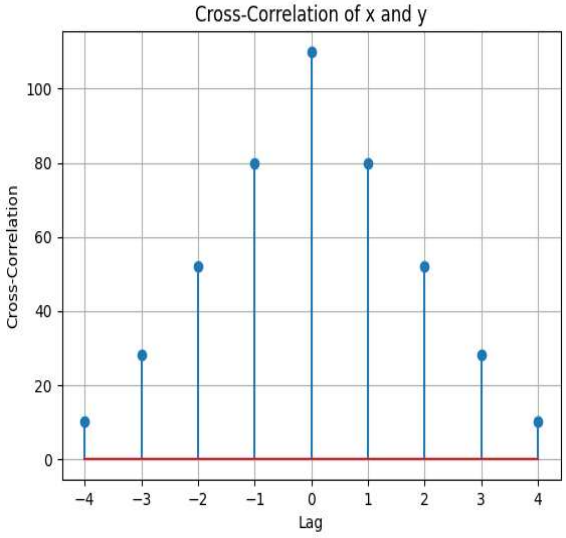
Auto-correlation measures the similarity of a sequence with itself. Given a sequence x[n], the auto-correlation is computed as: $r_{xx}[k] = \sum(x[n] * x[n-k])$

Similar to cross-correlation, the resulting sequence r_xx[k] represents the similarity of the sequence x[n] at each lag k.

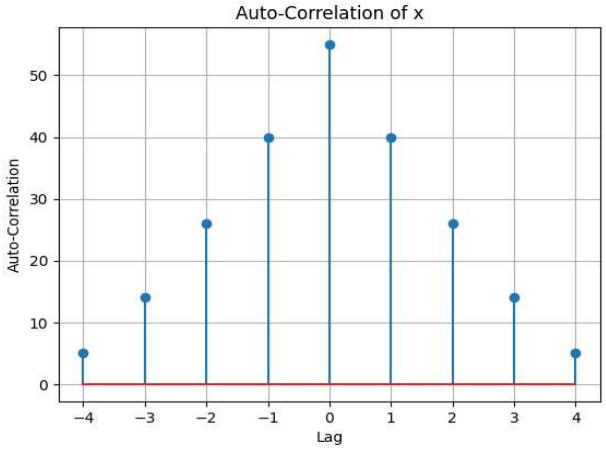
Python Code

(a) #Program to perform auto correlation and cross correlation

```
import numpy as np
import matplotlib.pyplot as plt
# First sequence
x = [1, 2, 3, 4, 5]
# Second sequence
y = [2, 4, 6, 8, 10]
# Perform cross-correlation
cross_corr = np.correlate(x, y, mode='full')
print('cross correlation',cross_corr)
Perform auto-correlation
auto_corr = np.correlate(x, x, mode='full')
print('autocorrelation', auto_corr)
# Plotting the results
```



```
lags = np.arange(-len(x) + 1, len(x))
plt.stem(lags, cross_corr)
plt.xlabel('Lag')
plt.ylabel('Cross-Correlation')
plt.title('Cross-Correlation of x and y')
plt.grid(True)
plt.show()
plt.stem(lags, auto_corr)
plt.xlabel('Lag')
plt.ylabel('Auto-Correlation')
plt.title('Auto-Correlation of x')
plt.grid(True)
plt.show()
```



#RESULT

cross correlation [10 28 52 80 110 80 52 28 10]

auto correlation [5 14 26 40 55 40 26 14 5]

Practice question:

3. Generate two sinusoidal signals using python and plot the autocorrelation and cross correlation of the same.
4. Generate signals having two frequencies 400 Hz and 4000 Hz using python and plot the autocorrelation of the signals.

Sl. No	**Criteria**	**Max Marks**	**Marks obtained**
Data sheet			
A	Problem statement	10	
B	Design & specifications	10	
C	Expected output	10	
Record			
D	Simulation/ Conduction of the experiment	15	
E	Analysis of the result	15	
	Viva	40	
	Total	100	
Scale down to 10 marks			

Experiment 7

AIM: Write a python code to extract features in the time domain for any signal.

Objective:

Extract time domain features from any audio signal (single tone/multitone)

Theory:

Feature extraction is the process of highlighting the most discriminating and impactful features of a signal. The evolution of features used in audio signal processing algorithms begins with features extracted in the time domain, which continue to play an important role in audio analysis and classification. Some of the basic features discussed here are:
i. read the audio file, ii. Measure the duration of the audio signal , iii. Amplitude variation of signal , iv.Spectrogram representation of Audio signal,v.MFCC, vi Chroma

i. #read audio file from google drive

```
import librosa   #librosa is a Python package for music and audio processing
import librosa.display   #
import IPython.display as ipd    #to play the audio signal
import matplotlib.pyplot as plt    #plot the audio signal in time domain
ipd.Audio('/content/drive/MyDrive/ColabNotebooks/trainingdatasets/cars001.wav)
#load a local WAV file
audio_path=('/content/drive/MyDrive/ColabNotebooks/trainingdatasets/cars001.wav')
x , sr = librosa.load(audio_path)   # 22050Hz
```

ii. #measure the duration of the audio signal

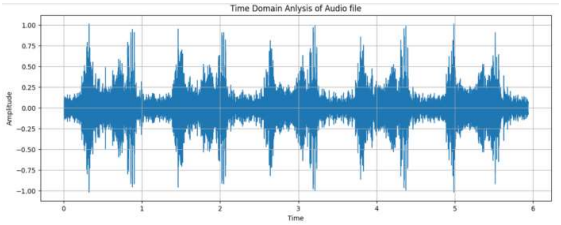
```
ipd.Audio('/content/drive/MyDrive/ColabNotebooks/trainingdatasets/cars001.wav')
#Duration of audio file
y=len(x)  #number of samples x
print(y)
#print(len(x))
y1=sr   #sampling frequency sr
print(y1)
duration_of_sound=y/y1  #number of samples/sampling frequency
print(duration_of_sound,"second")
```

Output: 5.9346938775510205 second

iii. Amplitude variation of signal in time domain

```
plt.figure(figsize=(14, 5))
#variable to change the x and y axix range
plt.grid(True )
```

```
#plotting the sampled signal
librosa.display.waveshow(x)
plt.xlabel("Time")
plt.ylabel("Amplitude")
```

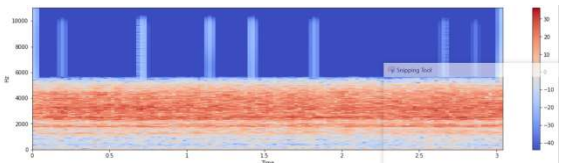



```
plt.title("Time Domain Anlysis of Audio file")
```

iv**. Spectrogram representation of Audio signal :**

A spectrogram is a visual representation of the spectrum of frequencies of a signal as it varies with time.

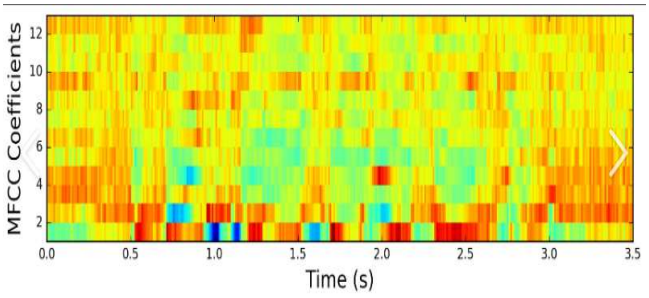
```
X = librosa.stft(x)
#converting into energy levels(dB)
Xdb = librosa.amplitude_to_db(abs(X))
plt.figure(figsize=(20, 5))
librosa.display.specshow(Xdb, sr=sr,
x_axis='time', y_axis='hz')
plt.colorbar()
```



v. Mel Frequency Cepstral Coefficients (MFCC)

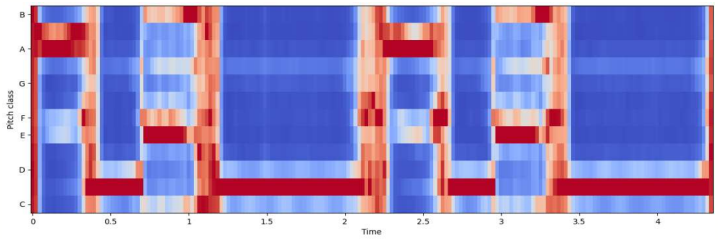
MFCC component : The **mfcc** function processes the entire speech data in a batch. Based on the number of input rows, the window length, and the overlap length, **mfcc** partitions the speech into 1551 frames and computes the cepstral features for each frame

```
librosa.load('/content/drive/MyDrive/ColabNotebooks/training datasets/cars001.wav')
mfccs = librosa.feature.mfcc(y=y, sr=sr, n_mfcc=40)
 #varable should be  y not x
plt.figure(figsize=(10,4))
librosa.display.specshow(mfccs, x_axis="time")
plt.colorbar()
plt.title('MFCC')
plt.tight_layout()
plt.show()
```



vi. **Chroma: We can use Chroma feature visualization to know how dominant the characteristics of a certain pitch {C, C♯, D, D♯, E, F, F♯, G, G♯, A, A♯, B} is present in the sampled frame.**

```
x, sr = librosa.load('/content/drive/MyDrive/Colab Notebooks/training datasets/cars014.wav')
hop_length = 512
S = np.abs(librosa.stft(y))
chroma = librosa.feature.chroma_stft(S=S, sr=sr)
plt.figure(figsize=(15, 5))
librosa.display.specshow(chroma, x_axis='time', y_axis='chroma', hop_length=hop_length, cmap='coolwarm')
```



Practice question:

1. Time domain analysis of various single tone signals and note the observations**.**

Sl. No	Criteria	Max Marks	Marks obtained
Data sheet			
A	Problem statement	10	
B	Design & specifications	10	
C	Expected output	10	
Record			
D	Simulation/ Conduction of the experiment	15	
E	Analysis of the result	15	
	Viva	40	
	Total	100	
Scale down to 10 marks			

Experiment 8

AIM: To write a python code to extract features in frequency domain for any signal.

Objective:

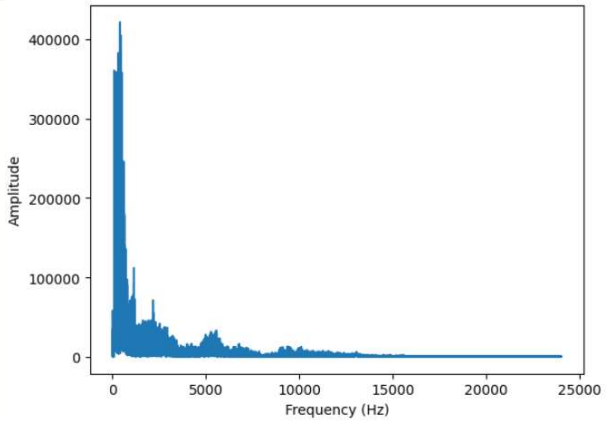
Extract frequency domain feature from any audio signal (single tone/multitone)

Theory:

The frequency domain-based audio feature extraction reveals the frequency content of the audio signals, band energy ratio, and so on. Some of the features analysed for the audio signals are:
i. measure the frequency of the signal using FFT, ii. mean, iii.standard deviation, vi skewness and kurtosis.

i. Read the audio file of ur choice

```
import scipy.io.wavfile as wavfile
import scipy
import scipy.fftpack as fftpk
import numpy as np
from matplotlib import pyplot as plt
s_rate, signal = wavfile.read('/content/drive/MyDrive/Colab Notebooks/training datasets/Copy of Sad(10).wav')
FFT = abs(fftpk.fft(signal))
freqs = fftpk.fftfreq(len(FFT), (1.0/s_rate))
plt.plot(freqs[range(len(FFT)//2)], FFT[range(len(FFT)//2)])
plt.xlabel('Frequency (Hz)')
plt.ylabel('Amplitude')
plt.show()
```



ii. #Measure of dispersion – variance – the second moment:

```
std = np.std(freqs)
maxv = np.amax(freqs)
```

```
minv = np.amin(freqs)
median = np.median(freqs)
iii. #Higher order moments – skewness and kurtosis:
skew = scipy.stats.skew(freqs)
kurt = scipy.stats.kurtosis(freqs)  # identify anomalies and outliers in many signal processing applications.
q1 = np.quantile(freqs, 0.25)
q3 = np.quantile(freqs, 0.75)
mode = scipy.stats.mode(freqs)[0][0]
iqr = scipy.stats.iqr(freqs)
print('mean as:', mean)
print('std as:',std)
print('shew as:',skew)
print('kurt as:',kurt)
print('q1 as:',q1)
```

Output:mean as: -5.206706237642656e-06

std as: 0.28867513457916105

shew as: 1.9759477879042912e-16

kurt as: -1.2000000002602538 q1 as: -0.25000260335311886

Practice questions:

1. Measure the frequency of various real time audio signals, compare the frequencies.

Sl. No	Criteria	Max Marks	Marks obtained
Data sheet			
A	Problem statement	10	
B	Design & specifications	10	
C	Expected output	10	
Record			
D	Simulation/ Conduction of the experiment	15	
E	Analysis of the result	15	
	Viva	40	
	Total	100	
Scale down to 10 marks			